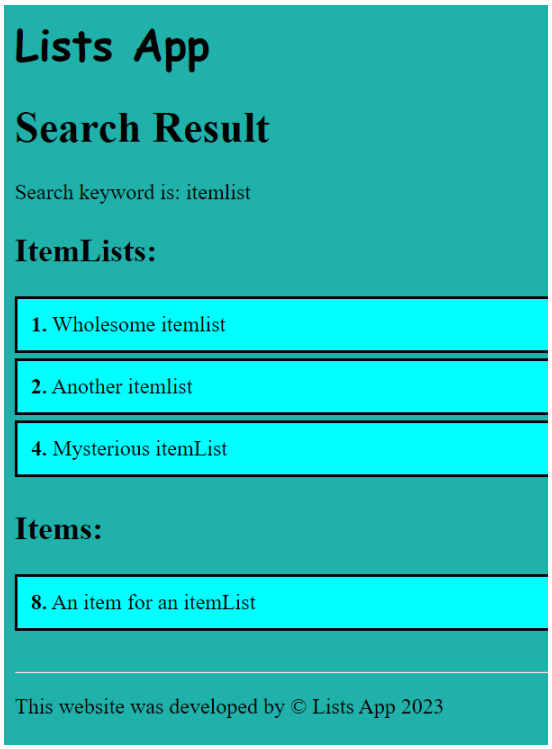
Object-Oriented Programming Coursework

Features of Lists App

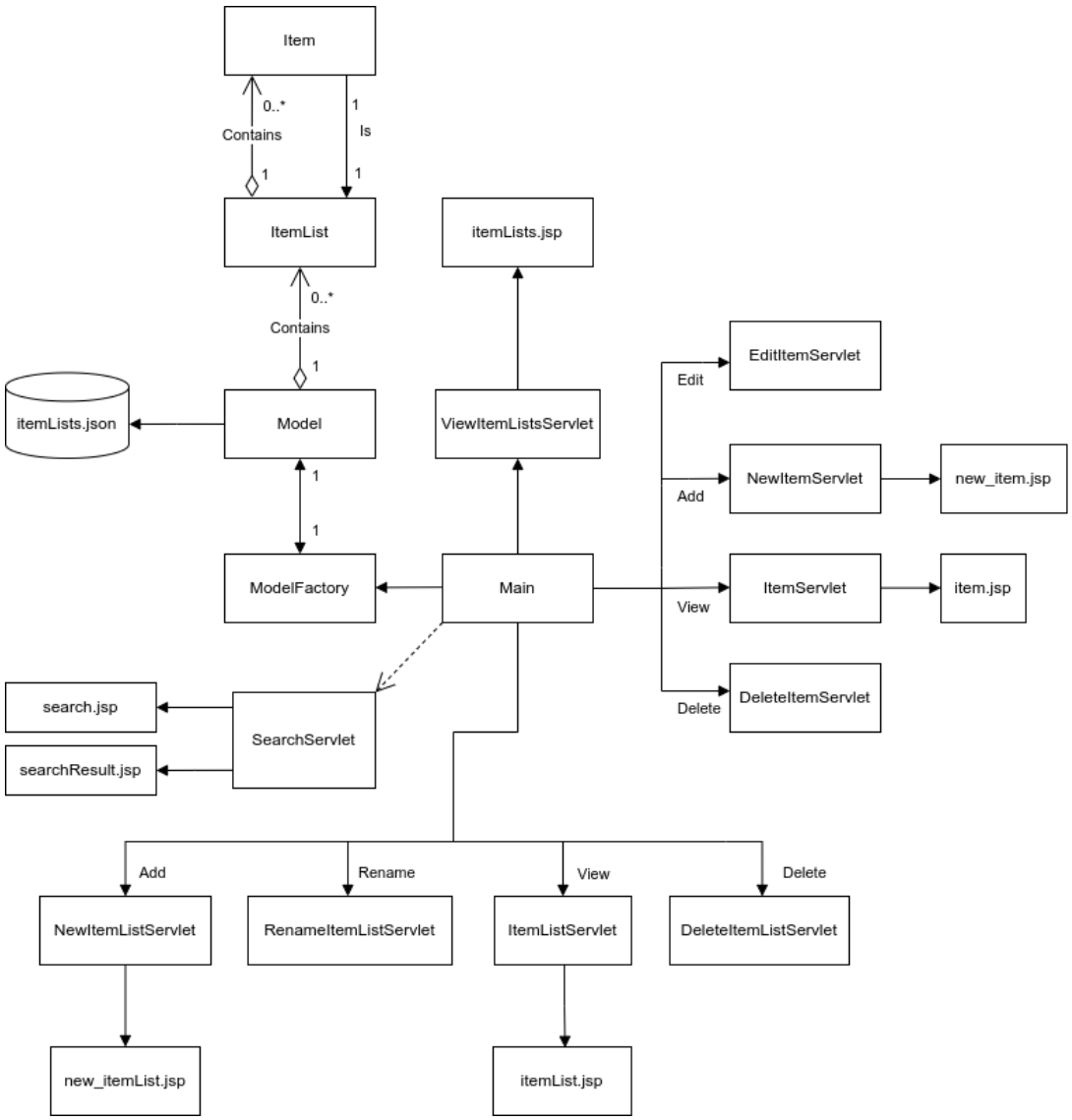
The Lists App is a Java-based project designed for Object-Oriented Programming coursework. The app is built using Apache Tomcat and Maven and allows the user to view a list of item-lists that are persisted via a JSON file. The user can perform several operations such as deleting, renaming, or creating new item-lists, and view each item-list. The app also allows the user to view and edit items within each item-list by adding, deleting, viewing, or editing them. Users can create new items and define them as links to other item-lists. In addition, there is a search option that allows users to search for keywords (case insensitive). Any item-list or item that contains the keyword substring will be displayed with a link to itself. This feature makes it easier for users to locate specific items or lists quickly. JavaScript is used only for extracting information from the DOM. Whilst it is not directly possible to upload images, users can insert HTML code for images that will be correctly displayed. The icon of the app can be clicked as a shortcut for returning to the home page.

The item-lists page

The search results page



UML class diagram for Lists App



Design and Programming Choices of Lists App

The Single Responsibility Principle was effectively applied when designing the List App. Each class is designed to perform a specific task, independently from the others, resulting in a well-structured and cohesive class design. The project successfully implemented Requirement 4 by allowing items to be pre-set as links to other item-lists, enabling users to have lists of lists without the need for complex recursive structures. The Model-View-Controller (MVC) pattern was used to separate the data, user interface, and logic into different components. The servlets act as a Controller, interacting with the Model class to retrieve and

update data before forwarding requests to their corresponding Java Servlet Pages (the View) where the data is displayed.

The naming convention used in the project is succinct and appropriate for each class and function's purpose. Where possible, a function is generalised, aiming to reduce code repetition. For example, the saveToDatabase() function in the Model class is called by all the functions that represent CRUD operations. Recycling code with slight modifications on the servlets used for the ItemLists, adapting them for Items potentially makes the program more readable.

To maintain simplicity and avoid unnecessary complexity that could slow down the development process, the project did not use abstract classes or interfaces. Two data model classes were used in the model, Item and ItemList, both using encapsulation together with getter and setter methods to promote abstraction and account for defensive design, making the code easier to read and understand, and reducing the risk of unintended side effects caused by changes to the internal state of an object. The program assumes that the ID attribute of an Item or ItemList is immutable for the sake of abstraction, and as a result, setter methods for the ID are not provided, unlike most other attributes.

A JSON file was used for local file storage instead of CSV or txt files as it is more elegant and better suited to handle commas in the title of an item-list, preventing any mistaken value-separating commas. The Model class has employed exception handling when loading a file, making debugging more manageable. The View component was created using JSPs to provide dynamic HTML code, while JavaScript was focused and limited to a specific frontend task, allowing users to edit and rename item-lists and items easily. Finally, the simple design created using CSS made testing more efficient and engaging.

If the Lists App were to be further developed in the future, improving the scalability of the search results page would be a crucial task. The current implementation of the item section utilizes multiple nested loops, which may perform adequately with a small number of items and item-lists, but is not scalable when dealing with large amounts of data. To improve scalability, it may be necessary to consider using more efficient algorithms and data structures that can handle a larger volume of data more effectively. Another change that could be done is use a base class from which Item and ItemList could inherit their common properties and methods to reduce duplication.